2021 算法模板

算法基础/工具

代码初始化

```
//#include <bits/stdc++.h>
//#pragma GCC optimize(2)
#include <stdio.h>
#include <cstring>
#include <algorithm>
#include <vector>
#include <stack>
#include <queue>
#include <iostream>
#include <map>
#include <cmath>
#include <set>
#include <time.h>
```

```cpp
#include <stdlib.h>
#include <random>

#define go(i, l, r) for(ll i = (l), i##end = (ll)(r); i <= i##end; ++i)
#define god(i, r, l) for(ll i = (r), i##end = (ll)(l); i >= i##end; --i)
#define ios ios_base::sync_with_stdio(0),cin.tie(0),cout.tie(0)
#define debug_in  freopen("in.txt","r",stdin)
#define debug_out freopen("out.txt","w",stdout);
#define pb push_back
#define all(x) x.begin(),x.end()
#define fs first
#define sc second
using namespace std;
typedef long long ll;
typedef unsigned long long ull;
typedef __int128 intt;
typedef pair<ll,ll> pii;
const ll inf_int = 1e9+10;
const ll inf_ll = 1e17;

void read(int &x){
    scanf("%d",&x);
}
void read(ll &x){
    scanf("%lld",&x);
}
template<class H, class... T> void read(H& h, T&... t) {
    read(h);
    read(t...);
}

void pt(){ cout<<'\n';}
template<class H, class ... T> void pt(H h,T... t){ cout<<" "<<h;pt(t...);}

//-----------------------------------------
const int maxn = 1e6 + 10;
int T;
```

```
int main() {
    debug_in; debug_out;



    return 0;
}
```

离散化

```
int lisa[maxn];int tail = 0;
void init(){
    sort(lisa+1,lisa+tail+1);
    tail = unique(lisa+1,lisa+tail+1)-lisa-1;
}
int ind(int x){
    return lower_bound(lisa+1,lisa+tail+1,x) - lisa;
}
```

手写哈希表

```
struct HashSet {
    struct node {
        int k, v, nex;
    } buf[N];
    int h[N], tot, mod = 1000009;
    void insert(int x) {
        int pos = x % mod;
        for (int i = h[pos]; i; i = buf[i].nex) {
            if (buf[i].k == x) { buf[i].v++; return; }
        }
        buf[++tot] = { x, 1, h[pos] };
        h[pos] = tot;
    }
    int find(int x) {
        int pos = x % mod;
```

```
        for (int i = h[pos]; i; i = buf[i].nex) {
            if (buf[i].k == x) return buf[i].v;
        }
        return 0;
    }
}mp;
```

__int128 读入输出

```
void read(__int128 &x) {
    x = 0;
    char ch;
    int flag = 1;
    while (ch = getchar()) {
        if (ch == '-') flag = -1;
        if (ch >= '0' && ch <= '9') break;
    }
    x = ch-'0';
    while ((ch = getchar()) >= '0' && ch <= '9') {
        x = x*10 + ch-'0';
    }
    x *= flag;
}
void out(__int128 x) {
    if (x < 0) {
        x = -x;
        putchar('-');
    }
    if (x >= 10) out(x / 10);
    putchar(x % 10 +'0');
}
```

常见log

```
log10(2):0.301030
log10(3):0.477121
```

```
log10(5):0.698970
log10(7):0.845098
ln(2):0.693147
ln(3):1.098612
ln(5):1.609438
ln(7):1.945910
```

特殊变量

```
自然对数E 在cmath库中的M_E
派在 cmath库中的M_PI

求一个数x有多少位。 x = 10^(y) 则位数为 (int)log(10,x) + 1
```

二分搜索

ans存储二分的答案

```
int l = mi,r = mx,ans;
while(l<=r){
    int mid = (l+r)>>1;
    int area = judge(mid);
    if(area>=K) l = mid+1,ans = mid;
    else r = mid-1;
}
```

数学

博弈论sg函数

正解

这里先介绍一下 SG 函数（其实只需要知道结论就行）。

一个状态的 SG 函数值（后面简称 SG 值）是其所有后继状态（进行一次操作后所到达的状态） ---手动换行---

SG值 集合的 mex（最小不属于这个集合的非负整数，比如说 $\operatorname{mex}\{0,1,2\}=3, \operatorname{mex}\{1\}=0, \operatorname{mex}\{\varnothing\}=0$）。

形式化的来说 $SG_x=\operatorname{mex}\{SG_y \mid y \text{ 是 } x \text{ 的后继状态}\}$。

结论：

1. 若当前局面的 SG 值为 0，先手必败，否则先手必胜。
2. 若一个游戏是多个独立游戏组成的，那么当前局面的 SG 值是多个独立游戏 SG 值的异或和 （就是 C++ 里面的 ^ 运算符）。

取模

如果模数不是质数，可以将 a/b mod p 转换成 (a mod bp)/b,这个在a整除b的时候满足

定理

树的构造个数

```
1 Cayley公式：对于n个不同的节点，能够组成的无根树（原来是无向连通图或者是有标志节
  点的树）的种数是n^(n-1)种
2 有根树：n^(n-1)
```

杨辉三角

```
1 杨辉三角中第i行第j列的数字正是C(i,j)的结果,下标从0开始,
2 是 (a+b)^x 的各项系数
```

调和级数

```
1 f(x) = 1/1 + 1/2 + 1/3 + 1/4 + ... + 1/n = ln(x) + 一个常数伽马
```

数的组合

```
1 两个数x,y,若干个x和若干个y相加不能组合成的最大数是(x-1)*(y-1)-1(x-1)*(y-1)
```

-1,也就是说(x-1)*(y-1)(x-1)*(y-1)及之后的数都可以用ax+byax+by来表示(a>=0,b>=0)(a>=0,b>=0)

切比雪夫定理

```
对于所有大于1的整数n，至少存在一个质数p，符合n < p < 2n
```

素数估计

$\pi(x)$表示[0,x]中有多少个素数

$\pi(x) \approx \frac{x}{\ln(x)}$

多个数求gcd

$gcd(a,b) = gcd(a,b-a)$

对于多个数求gcd有 $gcd(a,b,c) = gcd(a,b-a,c-b)$

数学公式

海伦公式求三角形面积

$S = \sqrt{(p*(p-a)*(p-b)*(p-c))}$

公式中a，b，c分别为三角形三边长，p为半周长，S为三角形的面积。

判断一个数是否是质数

```
bool isprime(int a){
    if(a==1) return 0;
    if(a==2||a==3) return 1;
    if(a%6!=1 && a%6!=5) return 0;
    int temp=sqrt(a);
    for(int i=5;i<=temp;i+=6)
```

```
    {
        if(a%i==0||a%(i+2)==0) return 0;
    }
    return 1;
}
```

埃氏素数筛法

```
const int maxn = 1e6+10;
ll P[maxn],tail;
bool vis[maxn];

void init(){
    int len = (int)sqrt(maxn)+1;
    for(int i = 2;i<maxn;i++){
        if(!vis[i]){
            P[tail++] = i;//将质数保存起来
            for(ll j = i*i;j<maxn;j+=i){
                vis[j] = true;
            }
        }
    }
}
```

线性筛素数

```
bool vis[maxn];
int P[maxn/10],tail;

void initP(int N){
    for(int i = 2; i <=N; i++){
        if(!vis[i]) P[tail++] = i;
        for(int j = 0; P[j] <= N / i; j++){
            vis[P[j] * i] = true;
            if(i % P[j] == 0) break;
        }
```

```
    }
}
```

求约数

对比较大的数求约数，在2e9范围内，一个数最多约数个数不超过1600，质因数不超过10个。比普通的求约数快10倍

```
bool vis[maxn];
ll P[maxn],tail;
int yin[maxn],poww[maxn],cnt;
int yue[maxn],last;
void init(){
    for(ll i = 2;i<maxn;i++){
        if(!vis[i]){
            P[tail++] = i;
            for(ll j = i*i;j<maxn;j+=i){
                vis[j] = true;
            }
        }
    }
}

ll gcd(ll a,ll b){
    return !b?a:gcd(b,a%b);
}
void div(ll x){
    cnt = 0;
    for(int i = 0;i<tail && P[i]*P[i]<=x;i++){
        if(x%P[i] == 0){
            yin[cnt] = P[i];
            int coun = 0;
            while(x%P[i] == 0){
                coun++;
                x/=P[i];
            }
            poww[cnt++] = coun;
        }
    }
```

```
    if(x>1) yin[cnt] = x,poww[cnt++] = 1;
}
void DFS(int now,int num){ //当前枚举的位置，当前的约数值
    if(now >= cnt){
        yue[last++] = num;
    }else{
        int cur = num;
        for(int i = 0;i<=poww[now];i++){ //遍历0-最高次方
            DFS(now+1,cur);
            cur *= yin[now];
        }
    }
}
```

最大公约数与最小公倍数

```
ll gcd(ll a, ll b) { return b?gcd(b,a%b):a;}
ll lcm(ll a, ll b) { return a/gcd(a,b)*b;}
```

矩阵乘法

```
struct Mat
{
    int mat[33][33];
    Mat(){
        memset(mat,0,sizeof mat);
    }
    Mat operator *(const Mat &b)const {
        Mat res;
        for(int i = 0;i<=N;i++){
            for(int j = 0;j<=N;j++){
                for(int k = 0;k<=N;k++){
                    res.mat[i][j] += mat[i][k] * b.mat[k][j] % mo
d; res.mat[i][j]%=mod;
                }
            }
```

```
        }
        return res;
    }

};
Mat ksm(Mat M,int b){
    Mat res;
    memset(&res,0,sizeof res);
    for(int i = 0;i<=N;i++) res.mat[i][i] = 1;
    while(b){
        if(b&1) res = res * M;
        M = M*M;
        b>>=1;
    }
    return res;
}
```

欧拉函数

求某个数的欧拉函数值

```
ll euler(ll n){
    ll t=n;
    for(ll i=2;i*i<=n;i++){
        if(n%i==0){
            t-=t/i;
            while(n%i==0)
                n/=i;
        }
    }
    if(n>1) t-=t/n;
    return t;
}
```

欧拉函数值打表

```cpp
const int maxn = 1e6+10;
int E[maxn];
void init()
{
    E[1] = 1;
    for(int i=2;i<maxn;i++){
        if(!E[i])
            for(int j=i;j<maxn;j+=i){
                if(!E[j]) E[j]=j;
                E[j] = E[j]/i*(i-1);
            }
    }
}
```

欧拉函数与质数一次性筛

```cpp
int P[maxn],tail;//质数
bool vis[maxn];
int E[maxn]; //欧拉值
ll sum[maxn];//欧拉值前缀和
void initEP(int n)
{
    E[1] = 1;
    for (int i = 2; i <n; i ++ ){
        if (!vis[i]){
            P[tail ++ ] = i;
            E[i] = i-1;
        }
        for (int j = 0; P[j] * i <= n; j ++ ){
            vis[P[j] * i] = true;
            if (i % P[j] == 0){
                E[i * P[j]] = E[i] * P[j];
                break;
            }
            E[i * P[j]] = E[i] * (P[j] - 1);
        }
    }

```

```
    for (int i = 1; i <= n; i ++ ) sum[i] = sum[i - 1] + E[i];
}
```

exgcd解线性同于方程

ax+by = c

d保存gcd(a,b)的结果，x,y是c = gcd(a,b)时对应的解

x，y的解系：$\begin{cases} x = \frac{c}{d}x + \frac{a}{d} \\ y = \frac{c}{d}y - \frac{b}{d} \end{cases}$

方程右边的x,y是c = gcd(a,b)时对应的解

```
void ex_gcd(ll a, ll b, ll &d, ll &x, ll &y){
    if(!b){
        d = a, x = 1, y = 0;
        return;
    }
    ex_gcd(b, a % b, d, y, x);
    y -= x * (a / b);
}
```

EXGCD求逆元

ax+by = 1 (mod P) 只要a与p互质就可以求出，a关于p的逆元

```
void exgcd(int a,int b,int& d,int& x,int& y)
{
    if(!b) { d = a; x = 1; y = 0; }
    else{ exgcd(b, a%b, d, y, x); y -= x*(a/b); }
}

int inv(int a, int p)
{
    int d, x, y;
    exgcd(a, p, d, x, y);
    return d == 1 ? (x+p)%p : -1;
}
```

费马小定理求逆元(快速幂求逆元)

费马小定理：$a^{P-1} = 1(\mod P)$（P是质数，且a和P互质）

```
ll ksm(ll a ,ll b,ll P){
    ll res = 1;
    while(b){
        if(b&1) res = (res*a)%P;
        a = a*a%P;
        b>>=1;
    }
    return res;
}
```

调和级数

求n/1 + n/2 + n/3 + ... + n/k 前K项

```
ll solve(ll n,ll k){
    ll m=sqrt(n);
    ll ans=0;
    for(ll i=1;i<=k&&i<=m;i++){
        ans=(ans+n/i);
    }
    if(k<=m){
        return ans;
    }
    else{
        for(ll i=n/min(n,k);i<=m;i++){
            ans =(ans+ (n/i - n/(i+1)) * i);
        }
        if(k<n){
            ans-=(n/(n/k)-k)*(n/k);
        }
        if( n / m == m)
```

```
            ans -= m;
    }
    return ans;
}
```

卡特兰数

O(N^2)递推

```
int N;
ll f[maxn];
f[0] = 1;f[1] = 1;
for(int i = 2;i<=N;i++){
    for(int j = 0;j<=i-1;j++){
        f[i] += f[j] * f[i-1-j];
    }
}
```

求卡特兰数 (公式方式 C(2n,n)/(n+1))

```
ll f[maxn];
f[0]=f[1]=1;
for(int i=2;i<=n;i++) f[i]+=f[i-1]*(4*i-2)/(i+1);
```

分数计算

```
struct node
{
public:
    ll son,mu;
    ll gcd(ll a,ll b){
        return !b? a : gcd(b,a%b);
    }
    void init(){
        ll g = gcd(son,mu);
```

```
        son/=g,mu/=g;
    }
    bool operator < (const node &o){
        return son * o.mu < o.son * mu;
    }
    friend node operator + (node &a,node &b){
        node ans;
        if(a.son == 0) return b;
        if(b.son == 0) return a;
        ans.mu = a.mu * b.mu;
        ans.son = a.son * b.mu + a.mu * b.son;
        ans.init();
        return ans;
    }
    friend node operator /(node a,ll x){
        node ans = a;
        ans.mu *= x;
        ans.init();
        return ans;
    }
};
```

组合数

大的组合数，C(n,m)复杂度O(m)

```
struct AC{
    ll f[maxn];
    void init(){
        f[0] = 1;
        for(int i = 1;i<maxn;i++) f[i] = f[i-1] * i %mod;
    }
    ll ksm(ll a,ll b){
        ll ans = 1;
        while(b){
            if(b&1) ans = ans * a %mod;
            a = a * a%mod;
            b>>=1;
        }
```

```
        return ans;
    }
    ll C(ll n,ll m) {
        if(n < m) return 0;
        ll res = 1;
        for(int i=1; i<=m; i++) {
            ll a = (n+i-m)%mod;
            ll b = i%mod;
            res = res*(a*ksm(b,mod-2)%mod)%mod;
        }
        return res;
    }
    ll lucas(ll n,ll m){
        return m? C(n%mod,m%mod) * lucas(n/mod,m/mod) %mod : 1;
    }
}ac;
```

N^2 预处理出组合数，适合1e3范围内的组合数，查询O(1)

```
ll C[N][N];
void initC(){ //初始化组合数C
    for(int i = 0;i<=N;i++) C[i][0] = 1;
    for(int i = 1;i<=N;i++){
        for(int j = 1;j<=i;j++){
            C[i][j] = C[i-1][j]+C[i-1][j-1];
        }
    }
}
```

预处理n固定不变的组合数C(n,m)

迭代公式为：$C_n^k = C_n^{k-1} \times (n-k+1)/k$

```
ll C[1000010];
void init(){
    ll ck = 1; C[0] = 1;
```

```
    for(ll k = 1;k<=n;k++){
        ck = ck*(n-k+1)/k;
        C[k] = ck;
    }
}
```

初始化所有的组合数在模mod的情况下的值

根据公式：$c(n,m) = n!/((n-m)! \times m!)$ 预处理出阶乘和阶乘的逆元,然后按公式进行计算

```
const ll maxn = 1e6+10;
const ll mod = 1000000007;
ll f[maxn],invf[maxn];
void init(int N){
    f[0]=invf[0]=f[1]=invf[1]=1;
    for(int i=2;i<=N;i++) f[i]=f[i-1]*i%mod,invf[i]=(mod-mod/i)*i
nvf[mod%i]%mod;
    for(int i=2;i<=N;i++) invf[i]=invf[i-1]*invf[i]%mod;
}
ll C(ll n,ll m)
{
    return f[n]*invf[n-m]%mod*invf[m]%mod;
}
```

高斯消元普通版

```
double eps = 1e-6;
int N;
double a[110][110];//保存系数 N个未知数，N个方程，外加一列常数
//下标从0开始，0～N-1行，0～N列，第N列是常数
int gauss(){
    int c,r;
    for(c = 0,r = 0;c<N;c++){
        int t = r;
        for(int i = r;i<N;i++){
```

```
            if(abs(a[i][c]) > abs(a[t][c])){
                t = i;
            }
        }
        if(abs(a[t][c]) < eps) continue;

        for(int j = 0;j<=N;j++) swap(a[r][j],a[t][j]); // 交换两行
        for(int j = N;j>=0;j--) a[r][j] /= a[r][c]; // 当前行当前列置1

        for(int i = r+1;i<N;i++){ //把下面行，当前列都置0
            if(abs(a[i][c]) < eps) continue;
            for(int j = N;j>=c;j--){
                a[i][j] -= a[r][j] * a[i][c];
            }
        }
        r ++ ;
    }
    if(r<N){ //此时剩余行的系数全是0，然后看右边的常数。0 * x = b，如果b是0，则无穷多解，不是0，则无解
        for(int i = r;i<N;i++){
            if(abs(a[i][N]) > eps) return 0;//无解
        }
        return 0;//无穷多个解
    }
    //反向代入求解方程
    for(int i = N-1;i>=0;i--){
        for(int j = i+1;j<=N;j++){
            a[i][N] -= a[i][j] * a[j][N];
        }
    }
    return 1; //唯一解。第i个x的解就是a[i][n]
}
```

高斯消元异或版

```
int N;
int a[110][110];//下标从0开始，0～N-1行，0～N列，第N列是常数
```

```
int gauss(){
    int c,r;
    for(c = 0,r = 0;c<N;c++){
        int t = r;
        for(int i = r+1;i<N;i++){
            if(abs(a[i][c]) > abs(a[t][c])){
                t = i;
            }
        }
        for(int j = 0;j<=N;j++) swap(a[t][j],a[r][j]);
        if(a[r][c] == 0) continue;

        for(int i = r+1;i<N;i++){
            if(a[i][c] == 0) continue;
            for(int j = N;j>=c;j--){
                a[i][j] ^= a[r][j];
            }
        }
        r++;
    }
    if(r<N){
        for(int i = r;i<N;i++){
            if(a[i][N] == 1) return 0;
        }
        return -1;
    }else{
        for(int i = N-1;i>=0;i--){
            for(int j = i+1;j<=N;j++){
                if(a[i][j] == 1) a[i][N] ^= a[j][N];
            }
        }
        return 1;
    }
}
```

定每行第一个1为主元

```
int gauss(){
```

```
    for(int i = 0;i<N;i++) fre[i] = -1;
    int c,r;
    for(c = r = 0;c<N;c++){
        int t = r;
        for(int i = r+1;i<N;i++){
            if(a[i][c] == 1){
                t = i;break;
            }
        }
        if(a[t][c] == 0) continue;
        fre[c] = r;
        for(int j = 0;j<=N;j++) swap(a[r][j],a[t][j]);
        for(int i = r+1;i<N;i++){
            if(a[i][c] == 0) continue;
            for(int j = N;j>=c;j--){
                a[i][j] ^= a[r][j];
            }
        }
        r++;
    }
    out();
    return 1;
}
```

线性基

其实就是高斯消元的特殊版

```
int N,L;//L代表二进制的位数
ll a[maxn];
bool insert(ll x){
    for(int j = L;j>=0;j--){
        if((x>>j & 1LL) == 0) continue;
        if(a[j]){
            x ^= a[j];
            continue;
        }
        for(int k = j - 1;k>=0;k--){
            if((x>>k & 1LL)){
```

```
                x ^= a[k];
            }
        }
        for(int k = L;k>j;k--){
            if((a[k]>>j & 1LL)){
                a[k] ^= x;
            }
        }
        a[j] = x;
        return 1;
    }
    return 0;
}
```

区间线性基

查询中数组[l,r]中选一些数，能异或出来的最大值

```
const int BASE = 31,maxn = 5e5 + 10;
int val[maxn][BASE], pos[maxn][BASE];
inline void insert(int i, int x)
{
    int k = i, tmp;
    for (int j = BASE - 1; j >= 0; --j)
        val[i][j] = val[i - 1][j], pos[i][j] = pos[i - 1][j];
    for (int j = BASE - 1; j >= 0; --j)
        if (x >> j)
        {
            if (!val[i][j])
            {
                val[i][j] = x;
                pos[i][j] = k;
                break;
            }
            else
            {
                if (k > pos[i][j])
                {
                    tmp = k, k = pos[i][j], pos[i][j] = tmp;
```

```
                    tmp = x, x = val[i][j], val[i][j] = tmp;
                }
                x ^= val[i][j];
            }
        }
}
inline void init()
{
    for (int i = 1; i <= N; ++i)
        for (int j = BASE - 1; j >= 0; --j)
            val[i][j] = pos[i][j] = 0;
}
inline int query(int l, int r)
{
    int ans = 0;
    for (int j = BASE - 1; j >= 0; --j)
        if ((ans ^ val[r][j]) > ans && pos[r][j] >= l)
            ans ^= val[r][j];
    return ans;
}
```

数据结构

带删除元素的STL堆

```
struct Heap{
    priority_queue<ll>q1,q2;
    inline void push(ll x){q1.push(x);}
    inline void erase(ll x){q2.push(x);}
    inline void updata(){while(!q1.empty()&&!q2.empty()&&q1.top()==q2.top())q1.pop(),q2.pop();}
    inline ll top(){updata();return q1.top();}
}Q;
```

ST表

静态求区间最大值，基于倍增的思想，查询复杂度O(1),处理复杂度O(nlogn)

```
int N,M;
int arr[maxn],Log2[maxn];//原始数组，log2(x)数组
int f[maxn][20];//F[i][j]: arr[i~i+2^j-1]的最大值

void ST_init(){ //初始化所有长度为2^x的区间最大值
    for(int i = 1;i<=N;i++) Log2[i] = log(i)/log(2); //初始化log求值，之后O(1)取值
    for(int i = 1;i<=N;i++) f[i][0] = arr[i];
    int len = log(N)/log(2) +1;
    for(int j = 1;j<len;j++){
        for(int i = 1;i<=N-(1<<j)+1;i++){
            f[i][j] = max(f[i][j-1],f[i+(1<<(j-1))][j-1]);
        }
    }
}
int query(int l,int r){ //查询arr[l~r]区间的最值
    int k = Log2[r-l+1];
    return max(f[l][k],f[r-(1<<k)+1][k]);
}
```

马拉车算法

坑神的模板

```
string s;
int ans[maxn],str[maxn],lef[maxn];
//ans：每一点的回文半径，str：插#字符后的字符串 lef：以某个为左边界的最长回文子串长度
int build(const string &s){
    int n = s.length(), m = (n + 1)*2, ret = 0;
    str[0] = '$'; str[m] = '@'; str[1] = '#'; ans[1] = 1;
    for (int i = 1; i <= n; i++)
        str[i*2] = s[i - 1], str[i*2+1] = '#';
    ans[1] = 1;
    for (int r = 0, p = 0, i = 2; i < m; ++i){
        if (r > i) ans[i] = min(r - i, ans[p * 2 - i]);
        else ans[i] = 1;
        while(str[i - ans[i]] == str[i + ans[i]]) ++ans[i];
        if (i + ans[i] > r) r = i + ans[i], p = i;
```

```
        ret = max(ret, ans[i] - 1);
    }
        // 计算维护以每个位置为起点的最长回文串
        for (int i = 0; i <= m; i++) lef[i] = 0;
        for (int i = 2; i < m; i++) if (lef[i - ans[i] + 1] < i + 1) lef[i - ans[i] + 1] = i + 1;
        for (int i = 1; i <= m; i++) if (lef[i] < lef[i - 1]) lef[i] = lef[i - 1];
    return ret;//最长回文串的长度
}
 int mid(int x, bool odd){
     //求以x为中心的最长回文子串长度，若是求偶回文串，这里中心认为是中间靠左
        if (odd) return ans[(x + 1) << 1] - 1;
        return ans[(x + 1) << 1 | 1] - 1;
    }
int left(int x){
    //求以x为左端点的最长回文串的长度
    return lef[(x + 1) << 1] - ((x+1) << 1);
}
```

树状数组

一维树状数组

```
int tr[maxn];

int lowbit(int x){
    return x&-x; //返回只保留二进制的最后一个1之后的值
}
void add(int idx,int x){ //向下标为idx的元素+x，同时更新其所影响结点的前缀和
    for(int i = idx;i<=N;i += lowbit(i)) //这里的N是数组最后一个下标，有时候不一定是N
        tr[i] += x;
}

int query(int idx){ //求下标为idx的前缀和
    int sum = 0;
```

```
    for(int i = idx;i>=1;i -= lowbit(i))
        sum += tr[i];
    return sum;
}
```

二维树状数组 [单点修改、区间查询]

```
int N,M;//N*M的矩阵
int tr[1111][1111];
int lowbit(int x){
    return x&-x;
}
void add(int x,int y,int v){
    for(int i = x;i<=N;i += lowbit(i)){
        for(int j = y;j<=M;j += lowbit(j)){
            tr[i][j] += v;
        }
    }
}
ll pre_sum(int x,int y){ //查询(1,1)到(x,y)矩阵和
    ll sum = 0;
    for(int i = x;i>=1;i -= lowbit(i)){
        for(int j = y;j>=1;j -= lowbit(j)){
            sum += tr[i][j];
        }
    }
    return sum;
}
ll query(int x1,int y1,int x2,int y2){ //查询(x1,y2)到(x2,y2)的矩阵和
    return pre_sum(x2,y2) - pre_sum(x1-1,y2) - pre_sum(x2,y1-1) + pre_sum(x1-1,y1-1);
}
```

二维树状数组 [区间修改，单点查询]

```cpp
int tr[1010][1010];//NxN的矩阵

int lowbit(int x){
    return x&-x;
}
void add(int x,int y,int v){
    for(int i = x;i<=N;i += lowbit(i))
        for(int j = y;j<=N;j += lowbit(j))
            tr[i][j] += v;
}
int query(int x,int y){
    int sum = 0;
    for(int i = x;i>=1;i -= lowbit(i))
        for(int j = y;j>=1;j -= lowbit(j))
            sum += tr[i][j];
    return sum;
}
//给左上角(x1,y1) ，右下角(x2,y2)的矩阵+v
add(x1,y1,+v);
add(x1,y2+1,-v);
add(x2+1,y1,-v);
add(x2+1,y2+1,+v);
```

线段树

单点修改，区间查询

```cpp
struct node{
    int l,r;
    int mx;
}tr[maxn*4];

void pushup(int u){
    //更新u所管区间的最大值
    tr[u].mx = max(tr[u<<1].mx,tr[u<<1|1].mx);
}
void build(int u,int l,int r){
    //结点u所管理的范围是[l,r]
    tr[u] = {l,r};
```

```
    if(l == r) return ;
    int mid = l+r>>1;
    build(u<<1,l,mid);
    build(u<<1|1,mid+1,r);
}

int query(int u,int l,int r){
    //查询[l,r]中的最大值
    if(tr[u].l>=l && tr[u].r<=r) return tr[u].mx;
    int mxl = 0,mxr = 0,mid = tr[u].l+tr[u].r>>1;
    if(l<=mid) mxl = query(u<<1,l,r);//注意依然传[l,r]参数
    if(r>mid) mxr = query(u<<1|1,l,r);
    return max(mxl,mxr);
}
void modify(int u,int idx,int v){
    //将下标idx位置元素改成v
    if(idx == tr[u].l && idx == tr[u].r) tr[u].mx = v;
    else{
        int mid = tr[u].l+tr[u].r>>1;
        if(idx<=mid) modify(u<<1,idx,v);
        else modify(u<<1|1,idx,v);
        pushup(u);
    }
}

```

区间修改，区间查询

```
struct segtree{
    struct node{
        int l,r;
        ll sum,lazy;
    }tr[maxn*4];
    void pushup(node &cur,node &L,node &R){
        cur.sum = L.sum + R.sum;
    }
    void pushdown(node &cur,node &L,node &R){
        if(cur.lazy == 0) return ;
```

```
        L.lazy += cur.lazy;
        R.lazy += cur.lazy;
        L.sum += (ll)(L.r - L.l + 1) * cur.lazy;
        R.sum += (ll)(R.r - R.l + 1) * cur.lazy;
        cur.lazy = 0;
    }
    void build(int l,int r,int u = 1){
        tr[u] = {l,r};
        if(l != r){
            int mid = (l+r)>>1;
            build(l,mid,u*2);
            build(mid+1,r,u*2+1);
        }
    }
    void modify(int l,int r,int v,int u = 1){
        if(l <= tr[u].l && tr[u].r <= r){
            tr[u].lazy += v;
            tr[u].sum += (tr[u].r - tr[u].l +1) * v;
        }else{
            pushdown(tr[u],tr[u*2],tr[u*2+1]);
            int mid  = (tr[u].l + tr[u].r) >>1;
            if(l<=mid) modify(l,r,v,u*2);
            if(r>mid) modify(l,r,v,u*2+1);
            pushup(tr[u],tr[u*2],tr[u*2+1]);
        }
    }
    ll query(int l,int r,int u = 1){
        if(l <= tr[u].l && tr[u].r <= r){
            return tr[u].sum;
        }else{
            pushdown(tr[u],tr[u*2],tr[u*2+1]);
            int mid = (tr[u].l + tr[u].r)>>1;
            ll sum = 0;
            if(l<=mid) sum += query(l,r,u*2);
            if(r > mid) sum += query(l,r,u*2+1);
            return sum;
        }
    }

}tree;
```

区间加/乘 修改与查询

```
主要思想：两个标记add mul，add表示是已经乘了mul之后的add，当前的add和当前的mul是独立的

当前是add mul
当区间+c时候，-> add+c,mul
当区间*c时候，-> add*c,mul *c
```

```cpp
#include <stdio.h>
#include <iostream>
#include <queue>
#include <map>
#include <vector>
#include <algorithm>
#include <cstring>
using namespace std;
typedef long long ll;
const int maxn = 1e5+10;

int N,M,mod;
int a[maxn];
struct Seg{
#define lson u<<1
#define rson u<<1|1
    struct node{
        int l,r;
        ll add,mul,sum;
    }tr[maxn*4];
    void pushup(node &U,node& L,node& R){
        U.sum = (L.sum + R.sum)%mod;
    }
    void build(int u,int l,int r){
        tr[u] = {l,r,0,1,a[l]};
```

```
        if(l == r){
            return ;
        }else{
            int mid = (l+r)>>1;
            build(lson,l,mid);
            build(rson,mid+1,r);
            pushup(tr[u],tr[lson],tr[rson]);
        }
    }
    void pushdown(node &U,node &L,node &R){
        L.sum = (L.sum * U.mul%mod + (L.r - L.l + 1) * U.add%mod)
%mod;
        R.sum = (R.sum * U.mul%mod + (R.r - R.l + 1) * U.add%mod)
%mod;

        L.add = (L.add * U.mul %mod + U.add)%mod;
        R.add = (R.add * U.mul %mod + U.add)%mod;
        L.mul = (L.mul * U.mul)%mod;
        R.mul = (R.mul * U.mul)%mod;
        U.add = 0;
        U.mul = 1;
    }
    void add(int u,int l,int r,int v){
        if(l <= tr[u].l && tr[u].r <= r){
            tr[u].sum = (tr[u].sum + v * (tr[u].r-tr[u].l+1)%mod)
%mod;
            tr[u].add = (tr[u].add + v )%mod;
        }else{
            pushdown(tr[u],tr[lson],tr[rson]);
            int mid = (tr[u].l + tr[u].r)>>1;
            if(l <= mid) add(lson,l,r,v);
            if(r > mid) add(rson,l,r,v);
            pushup(tr[u],tr[lson],tr[rson]);
        }
    }
    void mul(int u,int l,int r,int v){
        if(l<= tr[u].l && tr[u].r <= r){
            tr[u].sum = (tr[u].sum) * v%mod;
            tr[u].mul = (tr[u].mul * v)%mod;
            tr[u].add = (tr[u].add * v)%mod;
```

```
        }else{
            pushdown(tr[u],tr[lson],tr[rson]);
            int mid = (tr[u].l + tr[u].r)>>1;
            if(l<=mid) mul(lson,l,r,v);
            if(r>mid) mul(rson,l,r,v);
            pushup(tr[u],tr[lson],tr[rson]);
        }
    }
    node query(int u,int l,int r){
        if(l <= tr[u].l && tr[u].r <= r){
            return tr[u];
        }else{
            pushdown(tr[u],tr[lson],tr[rson]);
            int mid = (tr[u].l + tr[u].r)>>1;
            if(r<=mid) return query(lson,l,r);
            else if(l>mid) return query(rson,l,r);
            else{
                node ans1 = query(lson,l,r);
                node ans2 = query(rson,l,r);
                node ans;
                pushup(ans,ans1,ans2);
                return ans;
            }
        }
    }
}seg;
```

线段树套线段树

矩阵单点修改，查询矩阵区域最值

```
struct seg
{
    #define lson u<<1
    #define rson u<<1|1

    struct sub_node{
        int l,r,mx;
    };
```

```
    struct node{
        int l,r;
        sub_node sub_tr[1010*4];
    }tr[210*4];

    void build_sub(int l,int r,int id,int u = 1){
        tr[id].sub_tr[u] = {l,r,-1};
        if(l == r) return ;
        int mid = (l+r)>>1;
        build_sub(l,mid,id,lson);
        build_sub(mid+1,r,id,rson);
    }
    void build(int l,int r,int sl,int sr,int u = 1){
        tr[u] = {l,r};
        build_sub(sl,sr,u);
        if(l == r) return ;
        int mid = (l+r)>>1;
        build(l,mid,sl,sr,lson);
        build(mid+1,r,sl,sr,rson);
    }
    void pushup_sub(int id,int u){
        tr[id].sub_tr[u].mx = max(tr[id].sub_tr[lson].mx , tr[id]
.sub_tr[rson].mx);
    }
    void modify_sub(int idx,int v,int id,int u = 1){
        sub_node &tr2 = tr[id].sub_tr[u];
        if(tr2.l == idx && tr2.r == idx){
            tr2.mx =  max(tr2.mx,v);
            return ;
        }else{
            int mid = (tr2.l + tr2.r)>>1;
            if(idx<=mid) modify_sub(idx,v,id,lson);
            else modify_sub(idx,v,id,rson);
            pushup_sub(id,u);
        }
    }
    void modify(int idx1,int idx2,int v,int u = 1){ 给定坐标(x,y),
修改成v
        modify_sub(idx2,v,u);
        if(tr[u].l == tr[u].r) return ;
```

```
        int mid = (tr[u].l + tr[u].r)>>1;
        if(idx1 <= mid) modify(idx1,idx2,v,lson);
        else modify(idx1,idx2,v,rson);
    }
    int query_sub(int l,int r,int id,int u = 1){
        sub_node &tr2 = tr[id].sub_tr[u];
        if(l<= tr2.l && tr2.r <= r){
            return tr[id].sub_tr[u].mx;
        }else{
            int mid = (tr2.l + tr2.r)>>1;
            if(r<= mid) return query_sub(l,r,id,lson);
            else if(l > mid) return query_sub(l,r,id,rson);
            else return max(query_sub(l,r,id,lson),query_sub(l,r,
id,rson));
        }
    }

    int query(int l,int r,int sl,int sr,int u = 1){ //给定查询区域
        if(l<= tr[u].l && tr[u].r <= r){
            return query_sub(sl,sr,u);
        }else{
            int mid = (tr[u].l + tr[u].r)>>1;
            if(r<=mid) return query(l,r,sl,sr,lson);
            else if(l>mid) return query(l,r,sl,sr,rson);
            else return max(query(l,r,sl,sr,lson), query(l,r,sl,s
r,rson));
        }
    }
}seg;
```

轻重链剖分

```
int N,M,R,mod;
int w[maxn];//
vector<int> adj[maxn];//存树
struct segtree{
    struct node{
        int l,r;
```

```
7         ll sum,lazy;
8     }tr[maxn*4];
9     void pushup(node &cur,node &L,node &R){
10        cur.sum = L.sum + R.sum;
11        cur.sum %= mod;
12    }
13    void pushdown(node &cur,node &L,node &R){
14        if(cur.lazy == 0) return ;
15        L.lazy += cur.lazy; L.lazy %= mod;
16        R.lazy += cur.lazy; R.lazy %= mod;
17        L.sum += (ll)(L.r - L.l + 1) * cur.lazy; L.sum %= mod;
18        R.sum += (ll)(R.r - R.l + 1) * cur.lazy; R.sum %= mod;
19        cur.lazy = 0;
20    }
21    void build(int l,int r,int u = 1){
22        tr[u] = {l,r};
23        if(l != r){
24            int mid = (l+r)>>1;
25            build(l,mid,u*2);
26            build(mid+1,r,u*2+1);
27        }
28    }
29    void modify(int l,int r,int v,int u = 1){
30        if(l <= tr[u].l && tr[u].r <= r){
31            tr[u].lazy += v; tr[u].lazy %= mod;
32            tr[u].sum += (tr[u].r - tr[u].l +1) * v; tr[u].sum %
  = mod;
33        }else{
34            pushdown(tr[u],tr[u*2],tr[u*2+1]);
35            int mid  = (tr[u].l + tr[u].r) >>1;
36            if(l<=mid) modify(l,r,v,u*2);
37            if(r>mid) modify(l,r,v,u*2+1);
38            pushup(tr[u],tr[u*2],tr[u*2+1]);
39        }
40    }
41    ll query(int l,int r,int u = 1){
42        if(l <= tr[u].l && tr[u].r <= r){
43            return tr[u].sum;
44        }else{
45            pushdown(tr[u],tr[u*2],tr[u*2+1]);
```

```
46              int mid = (tr[u].l + tr[u].r)>>1;
47              ll sum = 0;
48              if(l<=mid) sum += query(l,r,u*2), sum%=mod;
49              if(r > mid) sum += query(l,r,u*2+1),sum%=mod;
50              return sum;
51          }
52      }
53
54  }tree;
55
56  struct treepou{
57      int tt = 0;
58      int tim[maxn],id[maxn],sz[maxn],hson[maxn],fa[maxn],top[maxn
    ],dep[maxn];
59      void init(){
60          dep[1] = 1;
61      }
62      void dfs1(int u){ //子树大小，每个节点的重孩子，每个孩子的父亲，节点的
    深度
63          sz[u] = 1;
64          for(auto v:adj[u]){
65              if(v == fa[u]) continue;
66              dep[v] = dep[u] + 1;
67              fa[v] = u;
68              dfs1(v);
69              sz[u] += sz[v];
70              if(sz[v] > sz[hson[u]]){
71                  hson[u] = v;
72              }
73          }
74      }
75      void dfs2(int u,int t){//弄出dfs序，每个节点的链头
76          tim[u] = ++tt;
77          id[tt] = u;
78          top[u] = t;
79
80          if(!hson[u]) return ;
81          dfs2(hson[u],t);
82          for(auto v:adj[u]){
83              if(v == fa[u] || v == hson[u]) continue;
```

```
            dfs2(v,v);
        }
    }
    void modify_tree(int x,int v){ //给根节点为x的子树的所有节点加v
        tree.modify(tim[x],tim[x]+sz[x]-1,v);
    }
    ll query_tree(int x){ //计算根节点为x的子树的点权和
        return tree.query(tim[x],tim[x]+sz[x]-1);
    }
    void modify_line(int x,int y,int v){//修改x到y的链上的点权，统一加1
        while(top[x] != top[y]){
            if(dep[top[x]] < dep[top[y]]) swap(x,y);
            tree.modify(tim[top[x]],tim[x],v);
            x = fa[top[x]];
        }
        if(dep[x] > dep[y]) swap(x,y);
        tree.modify(tim[x],tim[y],v);
    }
    ll query_line(int x,int y){ //查询x到y的链上的点权和
        ll ans = 0;
        while(top[x] != top[y]){
            if(dep[top[x]] < dep[top[y]]) swap(x,y);
            ans += tree.query(tim[top[x]],tim[x]); ans %= mod;
            x = fa[top[x]];
        }
        if(dep[x] > dep[y]) swap(x,y);
        ans += tree.query(tim[x],tim[y]); ans%=mod;
        return ans;
    }

}T;
void solve(){
    tree.build(1,N);
    T.init();
    T.dfs1(R);
    T.dfs2(R,R);
    for(int i =1;i<=N;i++) tree.modify(T.tim[i],T.tim[i],w[i]%mod);

```

```
    while(M--){
        int op,x,y,z;
        cin>>op;
        if(op == 1){
            cin>>x>>y>>z;
            T.modify_line(x,y,z);
        }else if(op == 2){
            cin>>x>>y;
            cout<<T.query_line(x,y)<<'\n';
        }else if(op == 3){
            cin>>x>>z;
            T.modify_tree(x,z);
        }else{
            cin>>x;
            cout<<T.query_tree(x)<<'\n';
        }
    }
}
int main(){

    cin>>N>>M>>R>>mod;
    for(int i =1;i<=N;i++) cin>>w[i];
    for(int i = 1;i<=N-1;i++){
        int x,y;cin>>x>>y;
        adj[x].pb(y);
        adj[y].pb(x);
    }
    solve();
    return 0;
}
```

平衡树treap

```
struct Treap{
    int root,idx;
    struct node{
        int l,r;
        int key,val;
```

```
6         int cnt,size;
7     }tr[100010];
8     void pushup(int p){
9         tr[p].size = tr[tr[p].l].size + tr[tr[p].r].size + tr[p].
  cnt;
10    }
11    int get_node(int key){
12        tr[++idx].key = key;
13        tr[idx].val = rand();
14        tr[idx].cnt = tr[idx].size = 1;
15        return idx;
16    }
17    void zig(int &p){
18        int q = tr[p].l;
19        tr[p].l = tr[q].r,tr[q].r = p,p = q;
20        pushup(tr[p].r),pushup(p);
21    }
22    void zag(int &p){
23        int q = tr[p].r;
24        tr[p].r = tr[q].l,tr[q].l = p,p = q;
25        pushup(tr[p].l),pushup(p);
26    }
27    void build(){
28        get_node(-inf);get_node(inf);
29        root = 1,tr[1].r = 2;
30        pushup(root);
31        if(tr[1].val < tr[2].val) zag(root);
32    }
33    void insert(int &p,int key){
34        if(!p) p = get_node(key);
35        else if(tr[p].key == key) tr[p].cnt++;
36        else if(key < tr[p].key){
37            insert(tr[p].l,key);
38            if(tr[tr[p].l].val > tr[p].val) zig(p);
39        }else{
40            insert(tr[p].r,key);
41            if(tr[tr[p].r].val > tr[p].val) zag(p);
42        }
43        pushup(p);
44    }
```

```
    void remove(int &p,int key){
        if(!p) return ;
        if(tr[p].key == key){
            if(tr[p].cnt > 1) tr[p].cnt--;
            else if(tr[p].l || tr[p].r){
                if(!tr[p].r || tr[tr[p].l].val > tr[tr[p].r].val)
{
                    zig(p);
                    remove(tr[p].r,key);
                }else{
                    zag(p);
                    remove(tr[p].l,key);
                }
            }else p = 0;
        }else if(key < tr[p].key) remove(tr[p].l,key);
        else remove(tr[p].r,key);
        pushup(p);
    }
    int get_rank_by_key(int p,int key){
        if(!p) return 0;
        if(key < tr[p].key) return get_rank_by_key(tr[p].l,key);
        else if(key == tr[p].key) return tr[tr[p].l].size + 1;
        else return tr[tr[p].l].size + tr[p].cnt + get_rank_by_ke
y(tr[p].r,key);
    }
    int get_key_by_rank(int p,int rank){
        if(!p) return inf;
        if(rank <= tr[tr[p].l].size) return get_key_by_rank(tr[p]
.l,rank);
        else if(rank <= tr[tr[p].l].size + tr[p].cnt) return tr[p
].key;
        else return get_key_by_rank(tr[p].r, rank - tr[tr[p].l].s
ize - tr[p].cnt);
    }
    int get_prev(int p,int key){
        if(!p) return -inf;
        if(key <= tr[p].key) return get_prev(tr[p].l,key);
        else return max(tr[p].key,get_prev(tr[p].r,key));
    }
    int get_next(int p,int key){
```

```
        if(!p) return inf;
        if(key >= tr[p].key) return get_next(tr[p].r,key);
        else return min(tr[p].key,get_next(tr[p].l,key));
    }
}trp;
```

使用方法

```
trp.build()
操作复杂度均为log
trp.insert(trp.root,v); //插入v
trp.remove(trp.root,v); //删除v
printf("%d\n",trp.get_rank_by_key(trp.root,v)-1);//因为最开始放了一个-inf，和inf，这里排名会比实际的排名-1
printf("%d\n",trp.get_key_by_rank(trp.root,v+1));//查询的时候，要查询比实际排名+1
printf("%d\n",trp.get_prev(trp.root,v));//获取前驱节点key
printf("%d\n",trp.get_next(trp.root,v));//获取后继节点key
```

pb_ds库之平衡树

可允许重复数字的平衡树

注意：速度比普通的手写的平衡树慢3倍,以及因为是维护的元素是结构体，使用lower_bound和upper_bound速度非常慢，要使用order_of_key(int x)来代替upper_bound来使用，其功能是查询比x小的数有多少个

```
#include <bits/stdc++.h>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
using namespace std;


int N,op,x;
```

```cpp
struct Tree{
    int inf = 1e9,idx = 0;
    struct node{
        int x,tim;
        bool operator <(node o) const{
            if(x != o.x) return x < o.x;
            else return tim < o.tim;
        }
    };
    tree<node, null_type, less<node>, rb_tree_tag, tree_order_statistics_node_update> T;
    void insert(int x){
        T.insert({x,++idx});
    }
    bool have(int x){ //判断是否有x
        node cur = {x,0};
        int order = T.order_of_key(cur);
        if(order + 1 > (int)T.size()) return 0;
        auto it = T.find_by_order(order);
        if((*it).x != x) return 0;
        return 1;
    }
    void erase(int x){
        if(!have(x)) return ;
        node cur = {x,0};
        int order = T.order_of_key(cur);
        auto it = T.find_by_order(order);
        T.erase(it);
    }
    int getrank(int x){ //查询x的排名
        return T.order_of_key({x,0}) + 1;
    }
    int getbyrank(int rank){ //通过排名x的元素是谁
        if((int)T.size() < rank) return -inf;
        return (*T.find_by_order(rank-1)).x;
    }
    int getpre(int x){//查询比x小，且最大的数
        int rank = getrank(x);
        if(rank == 1) return -inf;
        return getbyrank(rank-1);
```

```
    }
    int getnex(int x){//查询比x大，且最小的数
        int rank = getrank(x+1);
        if(rank > (int)T.size()) return -inf;
        return getbyrank(rank);
    }
}tr;
int main()
{
    cin>>N;
    for(int i = 1;i<=N;i++){
        int op,x;scanf("%d %d",&op,&x);
        if(op == 1) tr.insert(x);
        if(op == 2) tr.erase(x);
        if(op == 3) printf("%d\n",tr.getrank(x));
        if(op == 4) printf("%d\n",tr.getbyrank(x));
        if(op == 5) printf("%d\n",tr.getpre(x));
        if(op == 6) printf("%d\n",tr.getnex(x));
    }

    return 0;
}
```

可重复数字平衡树版本2

如果题目保证每一步都是合法的，可以用此版本，更少的代码量

```
#include <bits/stdc++.h>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
using namespace std;

int N,op,x;
struct Tree{
    int idx = 0;
    struct node{
        int x,tim;
        bool operator <(node o) const{
```

```
            if(x != o.x) return x < o.x;
            else return tim < o.tim;
        }
    };
    tree<node, null_type, less<node>, rb_tree_tag, tree_order_statistics_node_update> T;
    void insert(int x){
        T.insert({x,++idx});
    }
    void erase(int x){
        int rank = T.order_of_key({x,0});
        auto it = T.find_by_order(rank);
        T.erase(it);
    }
    int getrank(int x){
        return T.order_of_key({x,0}) + 1;
    }
    int getbyrank(int x){
        return (*T.find_by_order(x-1)).x;
    }
    int getpre(int x){
        int rank = T.order_of_key({x,0});
        return (*T.find_by_order(rank-1)).x;
    }
    int getnex(int x){
        int rank = T.order_of_key({x+1,0});
        return (*T.find_by_order(rank)).x;
    }
}tr;
int main()
{
//  freopen("in.txt","r",stdin);
    cin>>N;
    for(int i = 1;i<=N;i++){
        scanf("%d %d",&op,&x);
        if(op == 1) tr.insert(x);
        if(op == 2) tr.erase(x);
        if(op == 3) printf("%d\n",tr.getrank(x));
        if(op == 4) printf("%d\n",tr.getbyrank(x));
        if(op == 5) printf("%d\n",tr.getpre(x));
```

```
        if(op == 6) printf("%d\n",tr.getnex(x));

    }

    return 0;
}
```

图论

求树的所有子树的重心

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<cmath>
#include<functional>
#include<string>
#include<algorithm>
#include<iostream>
#include<set>
#include<vector>
#include<queue>
using namespace std;
const int N=4*1e5+10;
const int inf=0x3f3f3f3f;
int e[N],ne[N],idx;
int h[N];
int res[N];
int res1[N];
int size[N];
int p[N];
void add(int a,int b){
    e[idx]=b,ne[idx]=h[a],h[a]=idx++;
}
void dfs(int u,int fa){
    size[u]=1;//初始个数为0
    res[u]=u;//初始化节点重心为自身，因为如果不能从子树重心转移过来，那这个树的重心就是自己
```

```
    int son=0;
    int i;
    for(i=h[u];i!=-1;i=ne[i]){
        int j=e[i];
        if(j==fa)
        continue;
        p[j]=u;//j的父亲是u
        dfs(j,u);
        size[u]+=size[j];//计算节点总数
        if(size[j]>size[son])// 计算哪个是重（zhong）子树
        son=j;
    }
    // 如果符合条件就转移
    if(size[son]*2-size[u]>=0){
        res[u]=res[son];
        while(size[u]-2*size[res[u]]>0)
        res[u]=p[res[u]];  //通过迭代的方式往上走一格
        if(size[u]==2*size[res[u]]&&res[u]!=u){
            res1[u]=p[res[u]];
        }
    }
}
int main(){
    int m,n;
    cin>>n;
    int i;
    memset(h,-1,sizeof h);
    for(i=0;i<n-1;i++){
        int a,b;
        scanf("%d%d",&a,&b);
        add(a,b);
        add(b,a);
    }
    p[1]=1;
    dfs(1,-1);
    for (int i = 1; i <= n; i ++){

        if(!res1[i])
        printf("%d ", min(res1[i], res[i]);
        printf("%d\n", max(res1[i], res[i]);
```

```
    }

}
```

带权并查集

```
ll fa[maxn],dis[maxn];
ll find(ll x){
    if(fa[x] != x){
        ll f = fa[x];
        fa[x] = find(fa[x]);
        dis[x] += dis[f];
    }
    return fa[x];
}
bool join(ll x,ll y,ll v){
    ll fx = find(x),fy = find(y);
    if(fx != fy){
        fa[fy] = fx;
        dis[fy] = dis[x] + v - dis[y];
    }else if(dis[y] - dis[x] != v){ //权值冲突
        return false;
    }
    return true;
}
```

树的直径

dp[u]： 表示以u作为开始，往子树走的最长链长度

```
int N;
vector<int> adj[maxn];
int ans = 0;
int dp[maxn];
void dfs(int u,int fa = -1){
    dp[u] = 1;
    int len1 = 0,len2 = 0;
```

```
    for(auto v:adj[u]){
        if(v == fa) continue;
        dfs(v,u);
        if(dp[v] > len1){
            len2 = len1;
            len1 = dp[v];
        }else if(dp[v] > len2){
            len2 = dp[v];
        }
    }
    ans = max(ans,len1 + len2);
    dp[u] += len1;
}
int main(){
    cin>>N;
    for(int i =1;i<=N;i++){
        int x,y;cin>>x>>y;
        adj[x].pb(y);
        adj[y].pb(x);
    }
    dfs(1);
    cout<<ans<<'\n';


    return 0;
}
```

求图的SCC

```
struct SCC
{
    static const int maxn = 1e5+10;
    int dfn[maxn],low[maxn],scc_id[maxn],scc_sz[maxn],scc,tim;
    int sk[maxn],top;bool in_sk[maxn];

    void do_scc(){
        for(int i = 1;i<=N;i++){ // N是节点个数
            if(!dfn[i]){
```

```
                tarjan(i);
            }
        }
    }
    void tarjan(int u){
        dfn[u] = low[u] = ++tim;
        sk[++top] = u,in_sk[u] = true;
        for(int i = h[u];i;i = ne[i]){ // 遍历原来的图
            int v = e[i];
            if(!dfn[v]){
                tarjan(v);
                low[u] = min(low[u],low[v]);
            }else if(in_sk[v]){
                low[u] = min(low[u],dfn[v]);
            }
        }
        if(low[u] == dfn[u]){
            ++scc;
            int v;
            do{
                v = sk[top--],in_sk[v] = false;
                scc_sz[scc]++;
                scc_id[v] = scc;
            }while(v != u);
        }
    }
    void init_dag(){ //构建DAG图
        for(int u = 1;u<=N;u++){
            for(int i = h[u];i;i = ne[i]){
                int v = e[i],ww = w[i];
                int idu = scc_id[u],idv = scc_id[v];
                if(idu != idv){
                    dag[idu].pb({idv,ww});
                    in[idv]++;
                }
            }
        }
    }
}tar;
```

2-SAT

```
int N,M;
int h[maxn],e[maxn],ne[maxn],idx;
int dfn[maxn],low[maxn],scc_id[maxn],scc,tim;
int sk[maxn],top;bool in_sk[maxn];
void add(int a,int b){
    e[++idx] = b;
    ne[idx] = h[a];
    h[a] = idx;
}
int opp(int x){ //取反操作，x为1，x+N为0
    if(x<=N) return x + N;
    else return x-N;
}
void tarjan(int u){
    dfn[u] = low[u] = ++tim;
    sk[++top] = u;in_sk[u] = 1;
    for(int i = h[u];i;i = ne[i]){
        int v = e[i];
        if(!dfn[v]){
            tarjan(v);
            low[u] = min(low[u],low[v]);
        }else if(in_sk[v]){
            low[u] = min(low[u],dfn[v]);
        }
    }
    if(low[u] == dfn[u]){
        ++scc;
        int v;
        do{
            v = sk[top--];in_sk[v] = 0;
            scc_id[v] = scc;
        }while(v != u);
    }
}
void solve(){
    for(int i = 1;i<=2*N;i++){
```

```
        if(!dfn[i]){
            tarjan(i);
        }
    }
    bool ok = 1;
    for(int i = 1;i<=N;i++){
        if(scc_id[i] == scc_id[opp(i)]) ok = 0;
    }
    if(!ok) puts("IMPOSSIBLE");
    else{
        puts("POSSIBLE");
        int tag = 1;
        for(int i = 1;i<=N;i++){
            if(tag) tag = 0;else putchar(' ');
            if(scc_id[i] < scc_id[opp(i)]) putchar('1');
            else putchar('0');
        }
    }
}
int main(){
    // debug_in;
    read(N,M);
    for(int i = 1;i<=M;i++){
        int x,a,y,b;read(x,a,y,b);
        if(a == 0) x = x + N;
        if(b == 0) y = y + N;
        add(opp(x),y);
        add(opp(y),x);
    }

    solve();

    return 0;
}
```

欧拉回路

```
int T;
```

```
int G[300][300];
int du[300];
int fa[300];
int road[maxn];int tail;
int find(int x){
    return x == fa[x] ? x : fa[x] = find(fa[x]);
}
void dfs(int i){
    for(int j = 1;j<=150;j++){
        if(G[i][j] == 0 || G[j][i] == 0) continue;
        G[i][j] = 0;
        G[j][i] = 0;
        dfs(j);
    }
    road[++tail] = i;//存回溯经过的点，最后的欧拉回路是倒着的，不能直接存dfs序，因为可能不连续
}
void solve(){
    int cnt = 0,odd = 0;
    for(int i = 1;i<=150;i++){
        if(du[i] == 0) continue;
        if(du[i]%2 == 1) odd++;
        if(find(i) == i) cnt++;
    }
    // pt(cnt,odd);
    if(cnt>1 || odd>2)  puts("No Solution");//不联通 或者 有多余两个点度数是奇数
    else{
        if(odd == 0){ //无奇点，就随便找一个
            for(int i = 1;i<=150;i++){
                if(du[i] == 0) continue;
                dfs(i);
                break;
            }
        }else{//从两个奇数中的一个出发
            for(int i = 1;i<=150;i++){
                if(du[i] == 0) continue;
                if(du[i]%2 == 1){
                    dfs(i);
                    break;
```

```
                }
            }
        }
        for(int i = tail;i>=1;i--){
            putchar(char(road[i]));
        }
    }
}
int main(){
    // debug_in;

    read(T);
    for(int i =1;i<=150;i++) fa[i] = i;
    for(int i = 1;i<=T;i+=1){
        char a,b; cin>>a>>b;
        G[a][b] = G[b][a] = 1;
        fa[find(a)] = find(b);
        du[int(a)]++;
        du[int(b)]++;
    }
    solve();


    return 0;
}
```

二分图最大匹配

```
int N,M,E;
vector<int> adj[maxn];
int match[maxn];
bool st[555];
bool find(int u){
    for(auto v:adj[u]){
        if(!st[v]){
            st[v] = true;
```

```
            if(match[v] == 0 || find(match[v])){
                match[v] = u;
                return true;
            }
        }
    }
    return false;
}
void solve(){
    int ans = 0;
    for(int i =1;i<=N;i++){
        memset(st,0,M+10);
        if(find(i)) ans++;
    }
    cout<<ans<<'\n';
}
int main(){
//    debug;
    ios;

    cin>>N>>M>>E;
    for(int i =1;i<=E;i++){
        int x,y;cin>>x>>y;
        adj[x].pb(y);
    }
    solve();

    return 0;
}
```

堆优化dij

```
struct node{
    int id,w;
    bool operator <(const node &o) const{
        return w > o.w;
    }
};
```

```cpp
priority_queue<node> q;
int N,M,S;
vector<pii> adj[maxn];
bool vis[maxn];
int dis[maxn];

void dij(int s){
    memset(dis,0x3f,sizeof dis);
    q.push({s,0});
    dis[s] = 0;
    while(q.size()){
        node cur = q.top();q.pop();
        int u = cur.id;
        if(vis[u]) continue;
        vis[u] = 1;
        for(auto p:adj[u]){
            int v = p.fs,w = p.sc;
            if(dis[u] + w < dis[v]){
                dis[v] = dis[u] + w;
                q.push({v,dis[v]});
            }
        }
    }
    for(int i = 1;i<=N;i++){
        printf("%d ",dis[i]);
    }
}
int main(){
//    debug;
//    ios;

    cin>>N>>M>>S;
    for(int i =1;i<=M;i++){
        int x,y,z;read(x),read(y),read(z);
        adj[x].pb({y,z});
    }
    dij(S);


    return 0;
```

```
47 }
```

zkw最小费用最大流

```
1  struct Edge{
2      int from, to, f, w;
3  }E[1000005];
4  int Hed[100005], Nex[1000005], ct=1, Cur[100005];
5  void add(int a, int b, int f, int w){  //加边
6      E[++ct].from=a, E[ct].to=b, E[ct].f=f, E[ct].w=w, Nex[ct]=Hed
   [a], Hed[a]=ct;
7      E[++ct].from=b, E[ct].to=a, E[ct].f=0, E[ct].w=-w, Nex[ct]=He
   d[b], Hed[b]=ct;
8  }
9  const int INF = 0x3f3f3f3f;
10 // mincostmaxflow
11 int n, m, s, t, maxflow, mincost, Dis[100005], F[100005];
12 bool SPFA(){  //最短路分层，从汇点向源点分层能保证DFS走的一定是最短路，不会
   浪费时间走错路
13     queue<int> Q; Q.push(s);
14     memset(Dis, INF, sizeof Dis);
15     Dis[s] = 0; int k;
16     while(!Q.empty()){
17         k = Q.front(); Q.pop();
18         F[k] = 0;
19         for(int i=Hed[k]; i; i=Nex[i]){
20             if(E[i].f && Dis[k]+E[i].w<Dis[E[i].to]){
21                 Dis[E[i].to] = Dis[k]+E[i].w;
22                 if(!F[E[i].to])
23                     Q.push(E[i].to), F[E[i].to] = 1;
24             }
25         }
26     }
27     return Dis[t] != INF;
28 }
29 int DFS(int k, int flow){
30     if(k == t){maxflow += flow; return flow;}  //达到汇点更新最大流
31     int sum = 0; F[k] = 1;  //F[]保证了当出现 0 费用边的时候不会出现两个
```

```
点之间来回跑的情况
    for(int i=Cur[k]; i; i=Nex[i]){
        if(!F[E[i].to] && E[i].f  && Dis[E[i].to]==Dis[k]+E[i].w)
{
            Cur[k] = i;  //当前弧优化
            int p = DFS(E[i].to, min(flow-sum, E[i].f));
            sum += p, E[i].f -= p, E[i^1].f += p, mincost += p*E[
i].w;  //更新费用
            if(sum == flow) break;
        }
    }
    F[k] = 0;
    return sum;
}
void Dinic(){
    while(SPFA()){
        memcpy(Cur, Hed, sizeof Hed);
        DFS(s, INF);
    }
}
```

字符串

tire树

```
int node[maxn][26],cnt[maxn],idx;//node保存所有的结点信息，里面存储的下
一个结点的下标idx，cnt保存结点被标记的次数
                          //idx和单链表中的idx效果一样，就是用于创建新结点
时++idx，使用idx所在的结点来存储新结点
void insert(string s){
    int p = 0;
    for(auto c:s){
        int id = c-'a';
        if(!node[p][id]) node[p][id] = ++idx;//如果没有对应的子结点，
就创建
        p = node[p][id];//移动到子结点
    }
    cnt[p]++;
```

```
}
int query(string s){
    int p = 0;
    for(auto c: s){
        int id = c-'a';
        if(!node[p][id]) return 0;//已经没有对应的子结点了，说明此字符串就没有出现过
        p = node[p][id];
    }
    return cnt[p];//返回出现的次数
}
```

tire 异或查询

```
const int maxn = 2e6 + 10;
int N;
int node[maxn][2],id[maxn],idx;
void insert(int x,int pos){
    int p = 0;
    for(int i = 31;i>=0;i--){
        int v = int((x>>i) & 1);
        if(!node[p][v]) node[p][v] = ++idx;
        p = node[p][v];
    }
}
int query(int x){ //查询当前数和字典中的哪一个数异或起来最大
    int p = 0,ans = 0;
    for(int i = 31;i>=0;i--){
        int v = int((x>>i) & 1);
        if(node[p][v^1]) {
            ans |= (1<<i);
            p = node[p][v^1];
        }else p = node[p][v];
    }
    return ans;
}
```

哈希

```
struct Hash
{
    ull h1[maxn],h2[maxn],p1[maxn],p2[maxn],P1 = 13331,P2 = 131,mod1 = 1e9+7,mod2 = 1e8+7;
    char s[maxn];int len;
    void init_pow(){
        p1[0] = 1,p2[0] = 1;
        h1[0] = 0,h2[0] = 0;
        for(int i = 1;i<maxn;i++){
            p1[i] = p1[i-1]*P1%mod1;
            p2[i] = p2[i-1]*P2%mod2;
        }
    }
    void init_hs(){
        len = strlen(s+1);
        for(int i = 1;i<=len;i++){
            h1[i] = (h1[i-1]*P1%mod1+s[i])%mod1;
            h2[i] = (h2[i-1]*P2%mod2+s[i])%mod2;
        }
    }
    pii geths(int l,int r){
        ull hs1 = (h1[r] - h1[l-1]*p1[r-l+1]%mod1 + mod1)%mod1;
        ull hs2 = (h2[r] - h2[l-1]*p2[r-l+1]%mod2 + mod2)%mod2;
        pii p = {hs1,hs2};
        return p;
    }
}H;
```